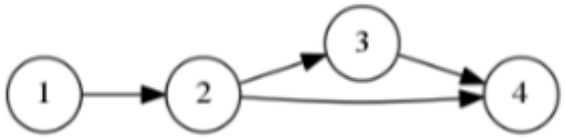
图论及其应用

hzwer,miskcoo

北京大学，清华大学

2019 年 10 月 4 日

图的矩阵存储



图可以直接用二维数组来存储。具体来说，如果一张图有 n 个结点，那么就使用 n × n 的二维数组来存储。

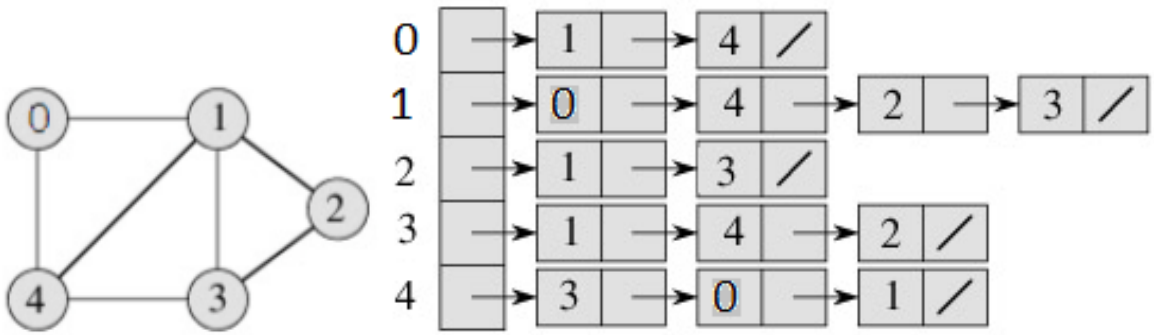
数组的每个元素的值代表了对应的边的数量。例如上面这张图就可以存储如下：

$$\begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

这种方式通常用来存储十分稠密的图，可以比较快地定位到某

图的链表存储

图还可以用邻接表来存储。也就是每个结点维护一个链表（vector），这个链表存储着以当前结点为开头的边。



通常情况下我们都是用这种方法来存储图的。

一些概念

一些概念

- **顶点的度**：在无向图中，某个顶点的度是与它相关联的边的数目。在有向图中，一个顶点的出度是以它为起始的边的数目，入度是以它为终止的边的数目。

一些概念

- **顶点的度**：在无向图中，某个顶点的度是与它相关联的边的数目。在有向图中，一个顶点的出度是以它为起始的边的数目，入度是以它为终止的边的数目。
- **简单路径**：顶点不重复的路径。

一些概念

- **顶点的度**：在无向图中，某个顶点的度是与它相关联的边的数目。在有向图中，一个顶点的出度是以它为起始的边的数目，入度是以它为终止的边的数目。
- **简单路径**：顶点不重复的路径。
- **自环**：从某个顶点出发连向它自身的边。

一些概念

- **顶点的度**：在无向图中，某个顶点的度是与它相关联的边的数目。在有向图中，一个顶点的出度是以它为起始的边的数目，入度是以它为终止的边的数目。
- **简单路径**：顶点不重复的路径。
- **自环**：从某个顶点出发连向它自身的边。
- **环**：从某个顶点出发再回到自身的路径，又称**回路**。

一些概念

- **顶点的度**：在无向图中，某个顶点的度是与它相关联的边的数目。在有向图中，一个顶点的出度是以它为起始的边的数目，入度是以它为终止的边的数目。
- **简单路径**：顶点不重复的路径。
- **自环**：从某个顶点出发连向它自身的边。
- **环**：从某个顶点出发再回到自身的路径，又称**回路**。
- **重边**：从一个顶点到另一个顶点有两条边直接相连。

一些概念

在无向图中，若从顶点 u 到 v 存在路径，那么称顶点 u 和 v 是**连通的**。如果无向图中任意一对顶点都是连通的，那么称此图为**连通图**。如果一个无向图不是连通的，则称它的一个极大连通子图为**连通分量**。这里的极大是指顶点个数极大。

拓扑排序

现在来考虑一个问题：现在有一个工程，这个工程被分成了很多部分。有一些部分要求前面某些部分完成后才可以开始进行。有些部分则可以同时进行。

拓扑排序

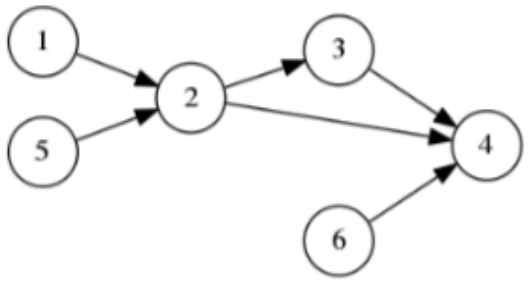
现在来考虑一个问题：现在有一个工程，这个工程被分成了很多部分。有一些部分要求前面某些部分完成后才可以开始进行。有些部分则可以同时进行。

我们可以把每个部分看作一个结点，刚刚这些限制看成是有向边。

拓扑排序

现在来考虑一个问题：现在有一个工程，这个工程被分成了很多部分。有一些部分要求前面某些部分完成后才可以开始进行。有些部分则可以同时进行。

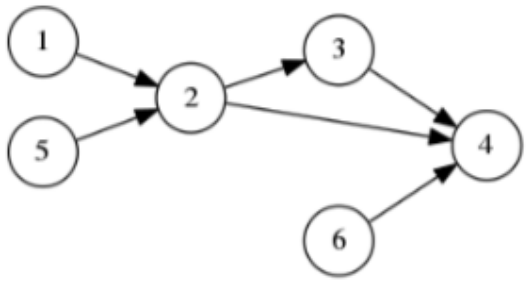
我们可以把每个部分看作一个结点，刚刚这些限制看成是有向边。



拓扑排序

现在来考虑一个问题：现在有一个工程，这个工程被分成了很多部分。有一些部分要求前面某些部分完成后才可以开始进行。有些部分则可以同时进行。

我们可以把每个部分看作一个结点，刚刚这些限制看成是有向边。



比如说这张图就可以看成是这样一个限制。这样的图有个特点：没有环！

因此这样的图也被成为有向无环图（DAG）。

拓扑排序

求出一个这个工程的工作序列的算法被成为**拓扑排序**。

比如说 1，5，2，3，6，4 就可以算作一个工作序列。

拓扑排序

求出一个这个工程的工作序列的算法被成为**拓扑排序**。

比如说 1，5，2，3，6，4 就可以算作一个工作序列。

拓扑排序的过程大概是这样的：

拓扑排序

求出一个这个工程的工作序列的算法被成为**拓扑排序**。

比如说 1，5，2，3，6，4 就可以算作一个工作序列。

拓扑排序的过程大概是这样的：

1. 选择一个入度为 0 的结点并直接输出。

拓扑排序

求出一个这个工程的工作序列的算法被成为**拓扑排序**。

比如说 1，5，2，3，6，4 就可以算作一个工作序列。

拓扑排序的过程大概是这样的：

1. 选择一个入度为 0 的结点并直接输出。
2. 删除这个结点以及与它关联的所有边。

拓扑排序

求出一个这个工程的工作序列的算法被成为**拓扑排序**。

比如说 1，5，2，3，6，4 就可以算作一个工作序列。

拓扑排序的过程大概是这样的：

1. 选择一个入度为 0 的结点并直接输出。
2. 删除这个结点以及与它关联的所有边。
3. 重复步骤 (1) 和 (2)，直到找不到入度为 0 的结点。

拓扑排序

求出一个这个工程的工作序列的算法被成为**拓扑排序**。

比如说 1，5，2，3，6，4 就可以算作一个工作序列。

拓扑排序的过程大概是这样的：

1. 选择一个入度为 0 的结点并直接输出。
2. 删除这个结点以及与它关联的所有边。
3. 重复步骤 (1) 和 (2)，直到找不到入度为 0 的结点。

通常情况下，在实现的时候会维护一个队列以及每个结点的入度。在删除边的时候顺便把相应结点的入度减去，当这个结点入度为 0 的时候直接将其加入队列。

例题 1

小明要去一个国家旅游。这个国家有 n 个城市，编号为 1 到 n，并且有 m 条道路连接着，小明准备从其中一个城市出发，并只往东走到城市 i 停止。

所以他就需要选择最先到达的城市，并制定一条路线以城市 i 为终点，使得线路上除了第一个城市，每个城市都在路线前一个城市东面，并且满足这个前提下还希望游览的城市尽量多。

现在，你只知道每一条道路所连接的两个城市的相对位置关系，但并不知道所有城市具体的位置。现在对于所有的 i，都需要你为小明制定一条路线，并求出以城市 i 为终点最多能够游览多少个城市。

其中 $1 \leq n \leq 100000, 1 \leq m \leq 200000$。

例题 1

题目中要求的是最长路

例题 1

题目中要求的是最长路

DAG

例题 1

题目中要求的是最长路

DAG

拓扑排序的过程中直接 DP

例题 2 CF698B

给出 n 个结点的父亲，问至少修改多少个结点的父亲，能使整张图变成一棵树（根的父亲为自己），要求输出任一方案。

其中 $1 \leq n \leq 200000$。

例题 2 CF698B

思考环和链的答案

例题 2 CF698B

思考环和链的答案

图的各个弱连通块是环 + 内向树，或者树/环。

例题 2 CF698B

思考环和链的答案

图的各个弱连通块是环 + 内向树，或者树/环。

先用拓扑排序把内向树消掉

例题 2 CF698B

思考环和链的答案

图的各个弱连通块是环 + 内向树，或者树/环。

先用拓扑排序把内向树消掉

剩下来的是一些环，每个环随便选一个结点当根，然后再把所有的根连在一起。

例题 2 CF698B

思考环和链的答案

图的各个弱连通块是环 + 内向树，或者树/环。

先用拓扑排序把内向树消掉

剩下来的是一些环，每个环随便选一个结点当根，然后再把所有的根连在一起。

答案是环数-（是否存在自环）

最小生成树

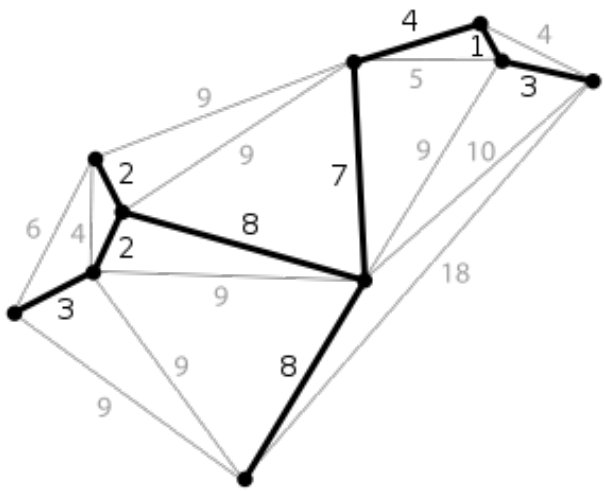
生成树：无向连通图 G 的一个子图如果是包含 G 的所有顶点的树，那么就称这个子图为 G 的生成树。

我们称生成树各边权值和为该树的**权**。对于无向连通图来说，权最小的生成树被成为**最小生成树**。

最小生成树

生成树：无向连通图 G 的一个子图如果是包含 G 的所有顶点的树，那么就称这个子图为 G 的生成树。

我们称生成树各边权值和为该树的**权**。对于无向连通图来说，权最小的生成树被成为**最小生成树**。



Kruskal 算法

Kruskal 算法是能够在 $\mathcal{O}(m \log m)$ 的时间内得到一个最小生成树的算法。它主要是基于贪心的思想：

Kruskal 算法

Kruskal 算法是能够在 $\mathcal{O}(m \log m)$ 的时间内得到一个最小生成树的算法。它主要是基于贪心的思想：

1. 将边按照边权从小到大排序，并建立一个没有边的图 T。

Kruskal 算法

Kruskal 算法是能够在 $\mathcal{O}(m \log m)$ 的时间内得到一个最小生成树的算法。它主要是基于贪心的思想：

1. 将边按照边权从小到大排序，并建立一个没有边的图 T。
2. 选出一条没有被选过的边权最小的边。

Kruskal 算法

Kruskal 算法是能够在 $\mathcal{O}(m \log m)$ 的时间内得到一个最小生成树的算法。它主要是基于贪心的思想：

1. 将边按照边权从小到大排序，并建立一个没有边的图 T。
2. 选出一条没有被选过的边权最小的边。
3. 如果这条边两个顶点在 T 中所在的连通块不相同，那么将它加入图 T。

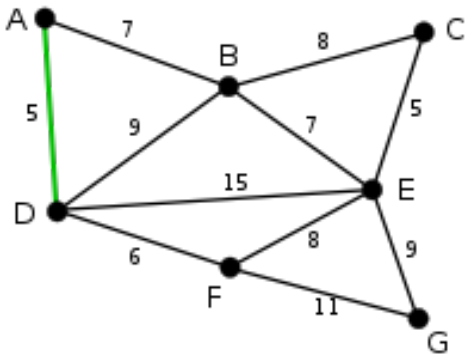
Kruskal 算法

Kruskal 算法是能够在 $\mathcal{O}(m \log m)$ 的时间内得到一个最小生成树的算法。它主要是基于贪心的思想：

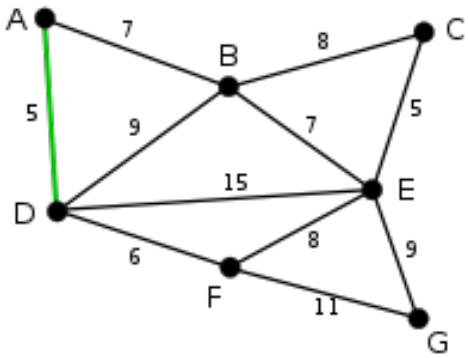
1. 将边按照边权从小到大排序，并建立一个没有边的图 T。
2. 选出一条没有被选过的边权最小的边。
3. 如果这条边两个顶点在 T 中所在的连通块不相同，那么将它加入图 T。
4. 重复（2）和（3）直到图 T 连通为止。

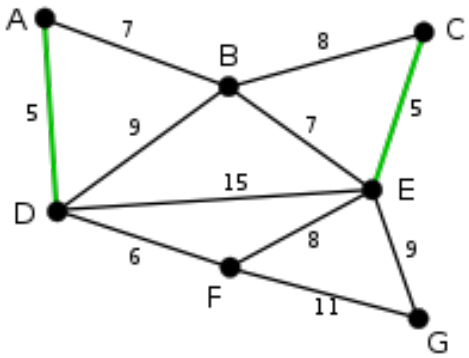
由于只需要维护连通性，可以不需要真正建立图 T，可以用并查集来维护。

Kruskal 算法

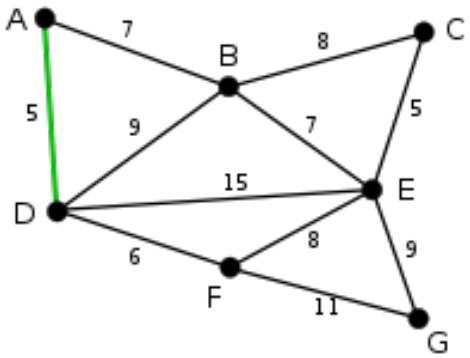


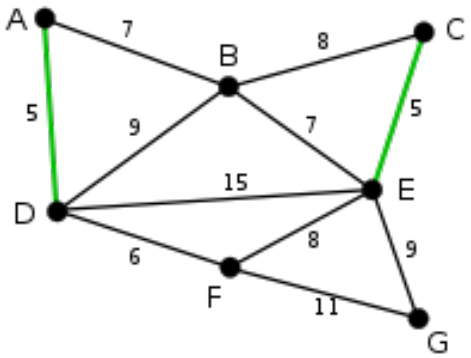
Kruskal 算法

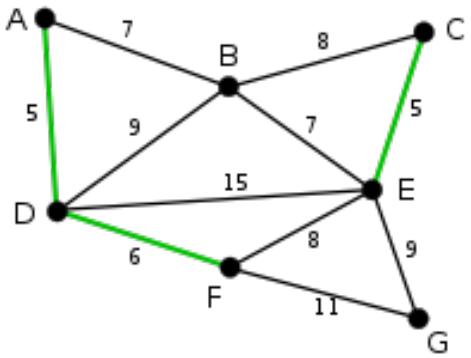




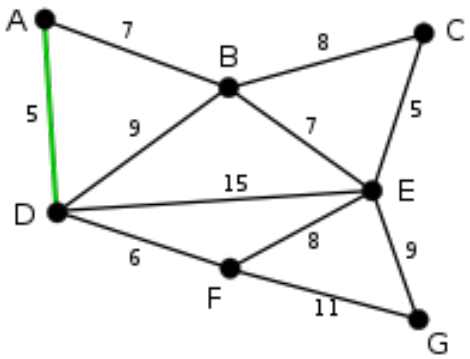
Kruskal 算法

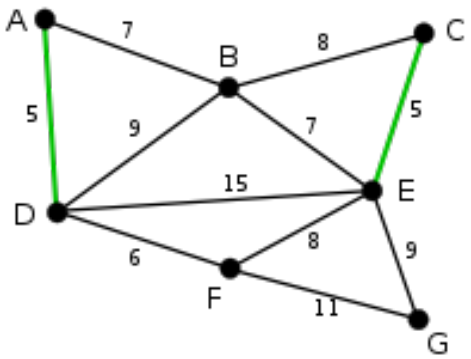

A
B
C
D
E
F
G
7
8
5
9
7
5
15
6
8
9
11

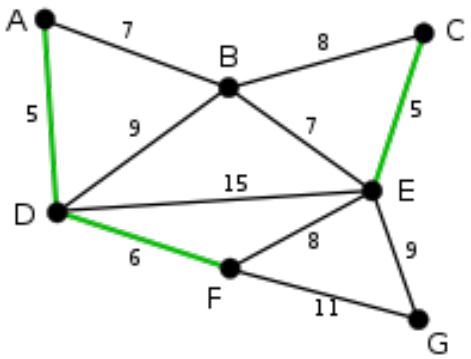

A
B
C
D
E
F
G
7
8
5
9
7
5
15
6
8
9
11

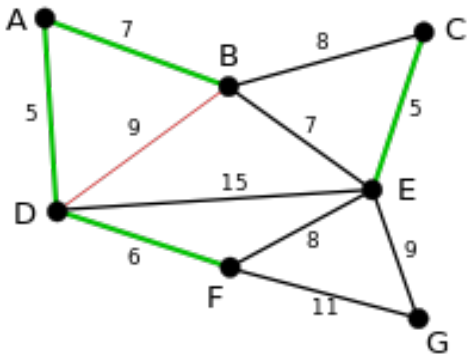

A
B
C
D
E
F
G
7
8
5
9
7
5
15
6
8
9
11

Kruskal 算法

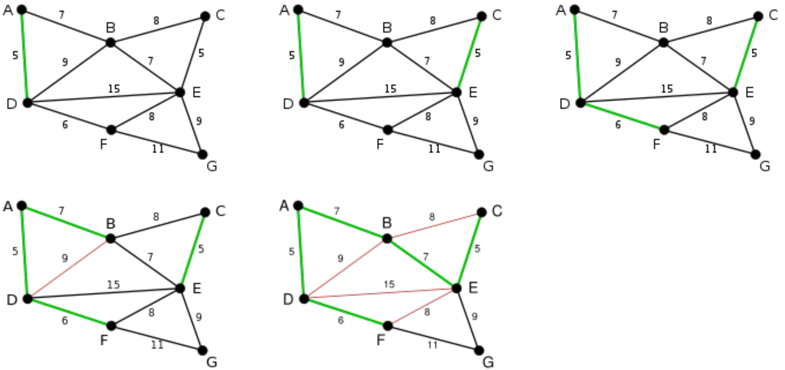

A
B
C
D
E
F
G
7
8
5
9
7
5
15
6
8
9
11


A
B
C
D
E
F
G
7
8
5
9
7
5
15
6
8
9
11

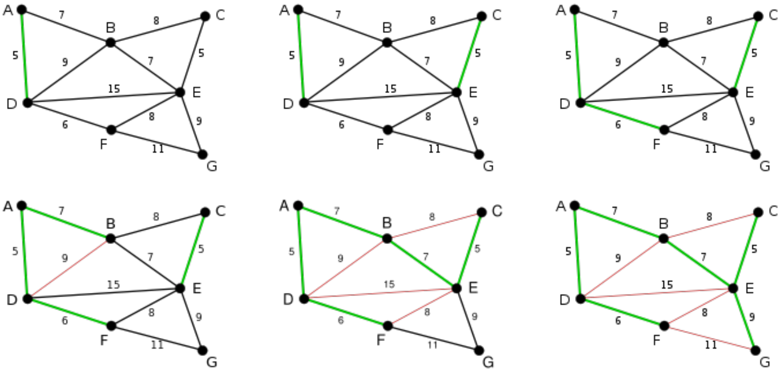

A
B
C
D
E
F
G
7
8
5
9
7
5
15
6
8
9
11


A
B
C
D
E
F
G
7
8
5
9
7
5
15
6
8
9
11

Kruskal 算法


A B C D E F G
7 8 5 9 7 5 15 6 8 9 11
A B C D E F G
7 8 5 9 7 5 15 6 8 9 11
A B C D E F G
7 8 5 9 7 5 15 6 8 9 11
A B C D E F G
7 8 5 9 7 5 15 6 8 9 11
A B C D E F G
7 8 5 9 7 5 15 6 8 9 11

Kruskal 算法


A B C D E F G 7 8 5 9 7 15 5 6 8 9 11
A B C D E F G 7 8 5 9 7 15 5 6 8 9 11
A B C D E F G 7 8 5 9 7 15 5 6 8 9 11
A B C D E F G 7 8 5 9 7 15 5 6 8 9 11
A B C D E F G 7 8 5 9 7 15 5 6 8 9 11
A B C D E F G 7 8 5 9 7 15 5 6 8 9 11

Prim 算法

Prim 算法和 Kruskal 算法一样也是寻找最小生成树的一种方法。

Prim 算法

Prim 算法和 Kruskal 算法一样也是寻找最小生成树的一种方法。

1. 先建立一个只有一个结点的树，这个结点可以是原图中任意的一个结点。

Prim 算法

Prim 算法和 Kruskal 算法一样也是寻找最小生成树的一种方法。

1. 先建立一个只有一个结点的树，这个结点可以是原图中任意的一个结点。
2. 使用一条边扩展这个树，要求这条边一个顶点在树中另一个顶点不在树中，并且这条边的权值要求最小。

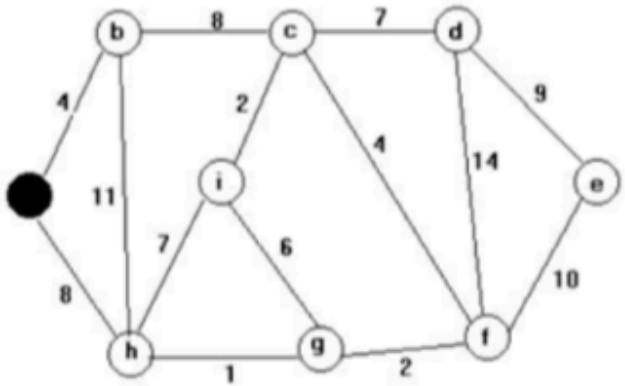
Prim 算法

Prim 算法和 Kruskal 算法一样也是寻找最小生成树的一种方法。

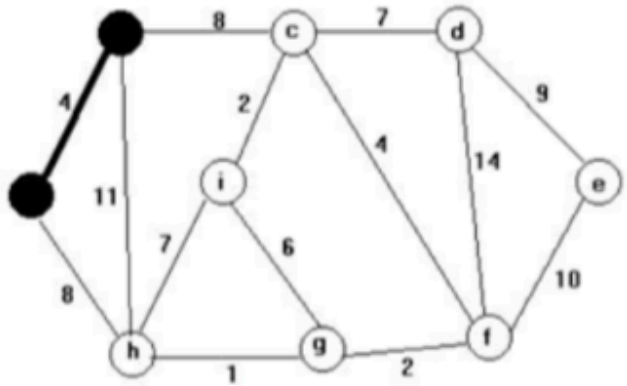
1. 先建立一个只有一个结点的树，这个结点可以是原图中任意的一个结点。
2. 使用一条边扩展这个树，要求这条边一个顶点在树中另一个顶点不在树中，并且这条边的权值要求最小。
3. 重复步骤（2）直到所有顶点都在树中。

Prim 算法

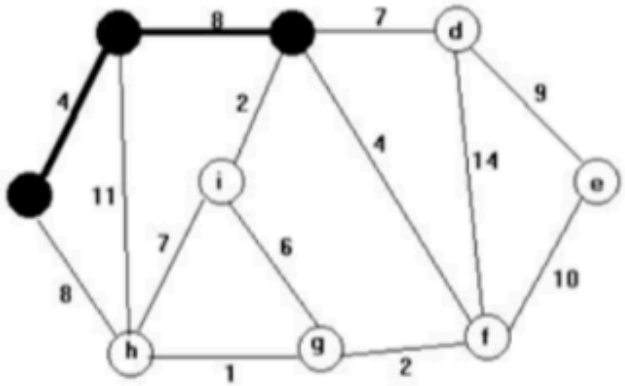
Iteration 0:



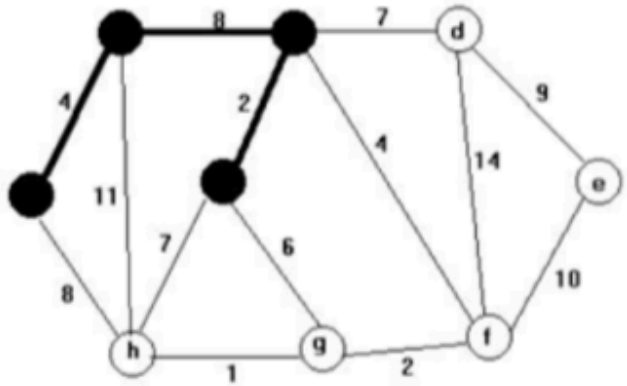
Iteration 1:



Iteration 2:

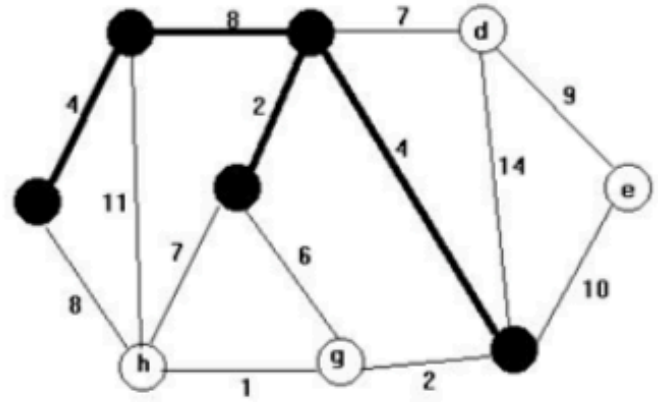


Iteration 3:

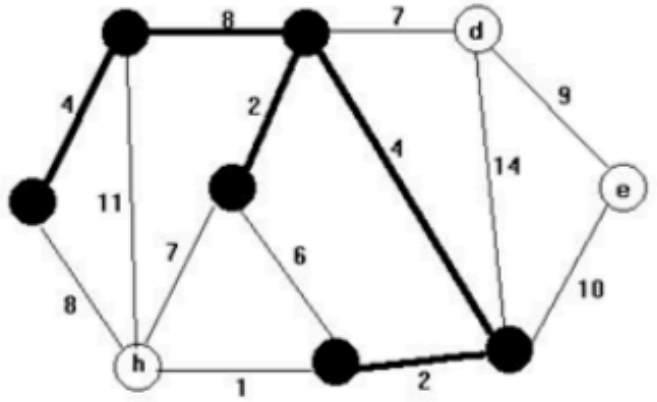


Prim 算法

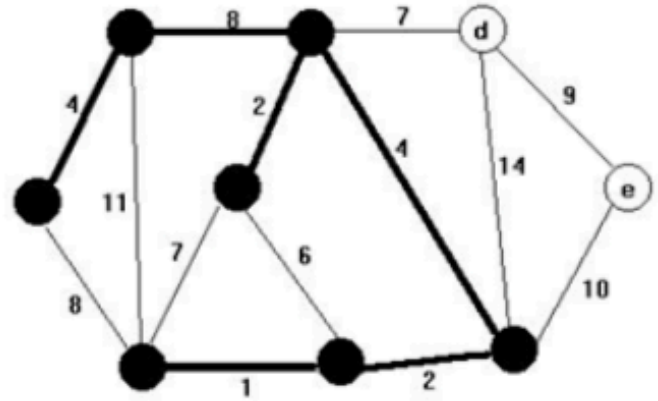
Iteration 4:



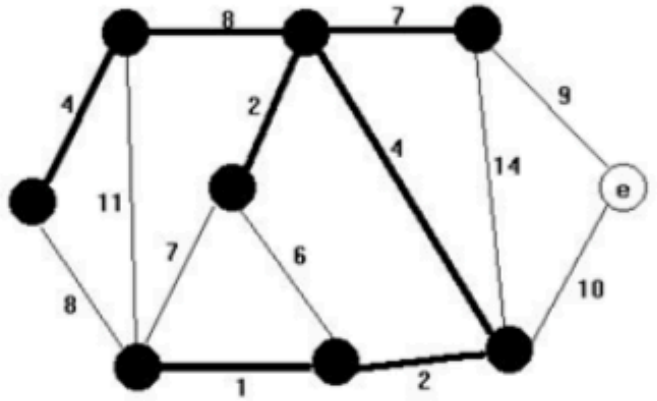
Iteration 5:



Iteration 6:

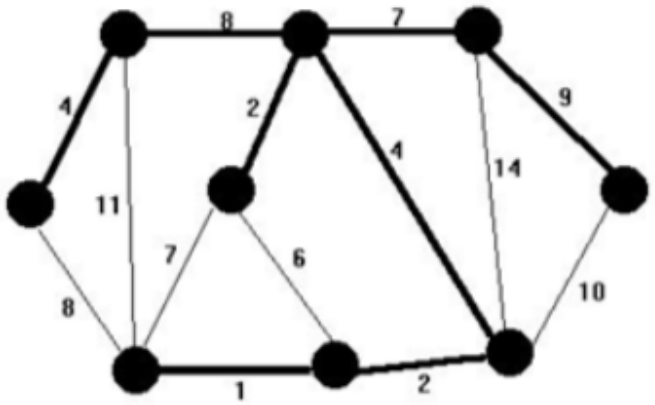


Iteration 7:

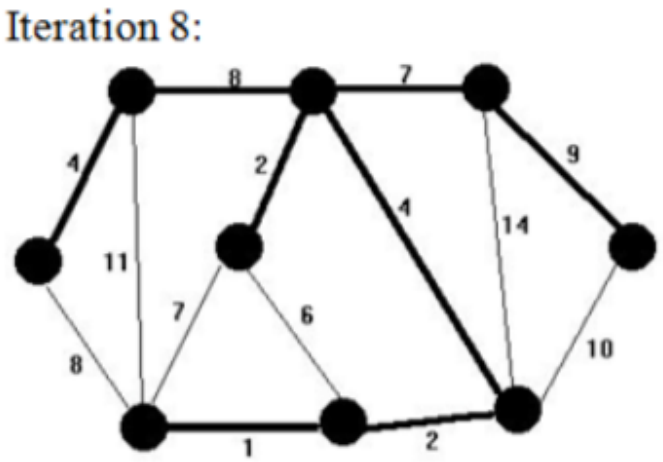


Prim 算法

Iteration 8:

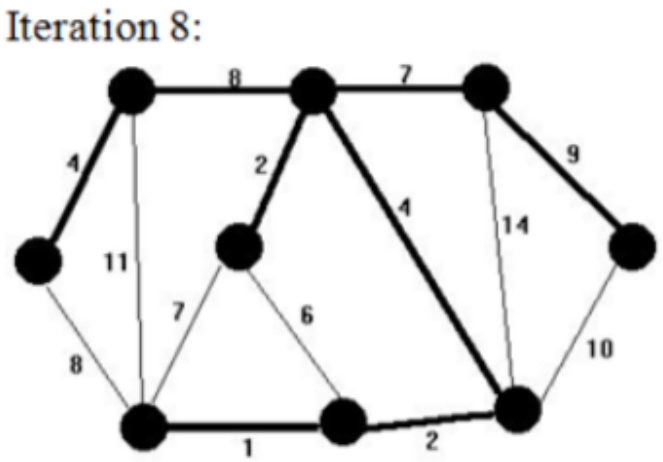


Prim 算法



具体实现的时候，可以为每个结点维护一个 key 值，表示它到已经选中的顶点中权值最小的边的权值。每次就在所有没有选中的顶点中找到 key 值最小的顶点，以及与它相关联的那条边加入树中并且更新与这个顶点相关联的所有顶点的 key 值。

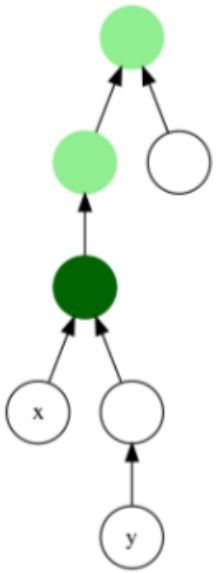
Prim 算法



具体实现的时候，可以为每个结点维护一个 key 值，表示它到已经选中的顶点中权值最小的边的权值。每次就在所有没有选中的顶点中找到 key 值最小的顶点，以及与它相关联的那条边加入树中并且更新与这个顶点相关联的所有顶点的 key 值。

这是可以用堆维护的，它的复杂度是 $\mathcal{O}(m \log n)$。

最近公共祖先



在一棵树上，两个结点的**最近公共祖先**（LCA）是它们的公共祖先中深度最大的那个顶点。

比如说，在左边的树中，顶点 x 和 y 的公共祖先用绿色标出，其中深绿色的顶点就是它们的最近公共祖先。

倍增算法

常见的求最近公共祖先的算法是**倍增算法**。

倍增算法

常见的求最近公共祖先的算法是**倍增算法**。

首先对于每个结点先进行 DFS 预处理出它的深度，再记录下它们往父亲方向走 $2^0, 2^1, 2^2, \cdots, 2^k$ 步所到达的结点。在这里 2^k 大于整棵树的最大深度。

倍增算法

常见的求最近公共祖先的算法是**倍增算法**。

首先对于每个结点先进行 DFS 预处理出它的深度，再记录下它们往父亲方向走 $2^0, 2^1, 2^2, \cdots, 2^k$ 步所到达的结点。在这里 2^k 大于整棵树的最大深度。

预处理完后，需要查询两个点 u 和 v 的 LCA 的时候，先将 u 和 v 中深度较大的一个利用先前处理出的数组走到和另一个结点相同的深度，这的所需要的操作次数不会超过 $\log_2 |\mathrm{depth}(u) - \mathrm{depth}(v)|$。

倍增算法

常见的求最近公共祖先的算法是**倍增算法**。

首先对于每个结点先进行 DFS 预处理出它的深度，再记录下它们往父亲方向走 $2^0, 2^1, 2^2, \cdots, 2^k$ 步所到达的结点。在这里 2^k 大于整棵树的最大深度。

预处理完后，需要查询两个点 u 和 v 的 LCA 的时候，先将 u 和 v 中深度较大的一个利用先前处理出的数组走到和另一个结点相同的深度，这的所需要的操作次数不会超过 $\log_2 |\mathrm{depth}(u) - \mathrm{depth}(v)|$。

接下来从 k 开始往下枚举，如果 u 和 v 如果往上走 2^i 后不相同，那么就将它们一起往上走这么多步。

倍增算法

常见的求最近公共祖先的算法是**倍增算法**。

首先对于每个结点先进行 DFS 预处理出它的深度，再记录下它们往父亲方向走 $2^0, 2^1, 2^2, \cdots, 2^k$ 步所到达的结点。在这里 2^k 大于整棵树的最大深度。

预处理完后，需要查询两个点 u 和 v 的 LCA 的时候，先将 u 和 v 中深度较大的一个利用先前处理出的数组走到和另一个结点相同的深度，这的所需要的操作次数不会超过 $\log_2 |\mathrm{depth}(u) - \mathrm{depth}(v)|$。

接下来从 k 开始往下枚举，如果 u 和 v 如果往上走 2^i 后不相同，那么就将它们一起往上走这么多步。

结束后如果 u 和 v 仍然不相等，再往上走一步。最后的顶点就是它们的 LCA。

倍增算法

miskcoo@bw:~/Code/oicode

```
4  int get_lca(int u, int v)
5  {
6      if(depth[u] < depth[v])
7          swap(u, v);
8      int d = depth[u] - depth[v];
9      for(int l = 0; d; ++l, d >>= 1)
10         if(d & 1) u = dist[l][u];
11
12     for(int p = MaxL - 1; u != v; p ? --p : 0)
13     {
14         if(dist[p][u] != dist[p][v] || p == 0)
15         {
16             u = dist[p][u];
17             v = dist[p][v];
18         }
19     }
20
21     return u;
22 }
```

倍增算法

倍增算法预处理的复杂度预处理是 $\mathcal{O}(n \log n)$，每次查询都是 $\mathcal{O}(\log n)$。

倍增算法

倍增算法预处理的复杂度预处理是 $\mathcal{O}(n \log n)$，每次查询都是 $\mathcal{O}(\log n)$。

不仅如此，我们可以动态地给树增加一些叶子结点。

倍增算法

倍增算法预处理的复杂度预处理是 $\mathcal{O}(n \log n)$，每次查询都是 $\mathcal{O}(\log n)$。

不仅如此，我们可以动态地给树增加一些叶子结点。

此外，在预处理的时候还可以顺便记录下这段路径的权值最大值，最小值或者权值和之类的信息，这样就可以在 $\mathcal{O}(\log n)$ 的时间内求出树上两点间路径权值的最大值、最小值还有权值和。

NOIP2013. 货车运输

A 国有 n 座城市，编号从 1 到 n，城市之间有 m 条双向道路。每一条道路对车辆都有重量限制，简称限重。现在有 q 辆货车在运输货物，每辆货车需要从一个城市运输到另一个城市。司机们想知道每辆车在不超过车辆限重的情况下，最多能运多重的货物。

其中 $0 < n < 10^4, 0 < m < 5 \times 10^4, 0 < q < 3 \times 10^4$

NOIP2013. 货车运输

A 国有 n 座城市，编号从 1 到 n，城市之间有 m 条双向道路。每一条道路对车辆都有重量限制，简称限重。现在有 q 辆货车在运输货物，每辆货车需要从一个城市运输到另一个城市。司机们想知道每辆车在不超过车辆限重的情况下，最多能运多重的货物。

其中 $0 < n < 10^4, 0 < m < 5 \times 10^4, 0 < q < 3 \times 10^4$

最大生成树

NOIP2013. 货车运输

A 国有 n 座城市，编号从 1 到 n，城市之间有 m 条双向道路。每一条道路对车辆都有重量限制，简称限重。现在有 q 辆货车在运输货物，每辆货车需要从一个城市运输到另一个城市。司机们想知道每辆车在不超过车辆限重的情况下，最多能运多重的货物。

其中 $0 < n < 10^4, 0 < m < 5 \times 10^4, 0 < q < 3 \times 10^4$

最大生成树、倍增算路径最小值

例题 1

给出一个有 n 个结点，m 条边的无向图，判断其最小生成树是否唯一。

其中 $0 < \mathtt{n} < 10^4, 0 < \mathtt{m} < 10^6$。

例题 1

给出一个有 n 个结点，m 条边的无向图，判断其最小生成树是否唯一。

其中 $0 < \mathtt{n} < 10^4, 0 < \mathtt{m} < 10^6$。

MST 什么时候唯一?

例题 1

给出一个有 n 个结点，m 条边的无向图，判断其最小生成树是否唯一。

其中 $0 < \mathtt{n} < 10^4, 0 < \mathtt{m} < 10^6$。

MST 什么时候唯一?

加入一条非树边会形成环

例题 1

给出一个有 n 个结点，m 条边的无向图，判断其最小生成树是否唯一。

其中 $0 < n < 10^4, 0 < m < 10^6$。

MST 什么时候唯一?

加入一条非树边会形成环

找到环上除了新加入的边外权值最大的边，该边权值小于这条非树边

Bellman-Ford 算法

给定了一个边带有权值（可以为负数）的有向图（不包含负环）和一个指定的顶点 s。要求求出从 s 到其余各点的最短路径长度。

Bellman-Ford 算法

给定了一个边带有权值（可以为负数）的有向图（不包含负环）和一个指定的顶点 s。要求求出从 s 到其余各点的最短路径长度。

Bellman-Ford 算法是一个比较直观的求解单源最段路问题的算法。

Bellman-Ford 算法

给定了一个边带有权值（可以为负数）的有向图（不包含负环）和一个指定的顶点 s。要求求出从 s 到其余各点的最短路径长度。

Bellman-Ford 算法是一个比较直观的求解单源最段路问题的算法。

我们可以肯定最短路径包含的边的条数不会超过 n-1 个，如果超过这个数，那么肯定形成了一个环，又因为这个环权值是正的，我们可以将路径上这个环删除，路径长度就会变小。

Bellman-Ford 算法

这个算法主要是构造一个最短路径长度数组的序列：dist[1][u], dist[2][u], ···, dist[n − 1][u]。

其中 dist[k][u] 表示从源 s 到 u **至多**经过 k 条边的最短路径的长度。

Bellman-Ford 算法

这个算法主要是构造一个最短路径长度数组的序列：dist[1][u], dist[2][u], ···, dist[n − 1][u]。

其中 dist[k][u] 表示从源 s 到 u **至多**经过 k 条边的最短路径的长度。

显然我们可以得到这样的关系：

Bellman-Ford 算法

这个算法主要是构造一个最短路径长度数组的序列：dist[1][u], dist[2][u], ···, dist[n − 1][u]。

其中 dist[k][u] 表示从源 s 到 u **至多**经过 k 条边的最短路径的长度。

显然我们可以得到这样的关系：

$$\texttt{dist}[1][\texttt{u}] = \texttt{w}[\texttt{s}][\texttt{u}]$$

Bellman-Ford 算法

这个算法主要是构造一个最短路径长度数组的序列：$\mathtt{dist}[1][u], \mathtt{dist}[2][u], \cdots, \mathtt{dist}[n-1][u]$。

其中 $\mathtt{dist}[k][u]$ 表示从源 s 到 u **至多**经过 k 条边的最短路径的长度。

显然我们可以得到这样的关系：

$$\mathtt{dist}[1][u] = w[s][u]$$

$$\mathtt{dist}[k][u] = \min(\mathtt{dist}[k-1][u], \min_{(v,u)\in E}(\mathtt{dist}[k-1][v] + w[v][u]))$$

Bellman-Ford 算法

这个算法主要是构造一个最短路径长度数组的序列：$\mathtt{dist}[1][u], \mathtt{dist}[2][u], \cdots, \mathtt{dist}[n-1][u]$。

其中 $\mathtt{dist}[k][u]$ 表示从源 s 到 u **至多**经过 k 条边的最短路径的长度。

显然我们可以得到这样的关系：

$$\mathtt{dist}[1][u] = w[s][u]$$

$$\mathtt{dist}[k][u] = \min(\mathtt{dist}[k-1][u], \min_{(v,u)\in E}(\mathtt{dist}[k-1][v] + w[v][u]))$$

由于每次计算 dist 数组复杂度都是 $\mathcal{O}(|E|)$ 的，总的复杂度就是 $\mathcal{O}(|V||E|)$。

SPFA 算法

事实上，你会发现我们每次进行的计算可以看成是一次“松弛”。假设我们现在已经得到了 Bellman-Ford 算法某个阶段的 dist 数组，然后我们发现了一条 s 到 u 的距离比 dist[u] 更加短的路径。我们更新了 dist[u]。

SPFA 算法

事实上，你会发现我们每次进行的计算可以看成是一次“松弛”。假设我们现在已经得到了 Bellman-Ford 算法某个阶段的 dist 数组，然后我们发现了一条 s 到 u 的距离比 dist[u] 更加短的路径。我们更新了 dist[u]。

那么接下来直接受到影响的就是与 u 直接关联的顶点 v，也就是如果 $\texttt{dist[u]} + \texttt{w[u][v]} < \texttt{dist[v]}$ 的话，s 到 v 的最短路就可以利用 s 到 u 的最短路加上 u 到 v 的边来更新。这样的话与 v 直接关联的顶点又会受到影响……不断这样持续下去直到最后没有顶点能被影响。

SPFA 算法

事实上，你会发现我们每次进行的计算可以看成是一次“松弛”。假设我们现在已经得到了 Bellman-Ford 算法某个阶段的 dist 数组，然后我们发现了一条 s 到 u 的距离比 dist[u] 更加短的路径。我们更新了 dist[u]。

那么接下来直接受到影响的就是与 u 直接关联的顶点 v，也就是如果 $\texttt{dist[u]} + \texttt{w[u][v]} < \texttt{dist[v]}$ 的话，s 到 v 的最短路就可以利用 s 到 u 的最短路加上 u 到 v 的边来更新。这样的话与 v 直接关联的顶点又会受到影响……不断这样持续下去直到最后没有顶点能被影响。

那么一个优化就是我们利用队列存储这些需要更新的结点，每次从队列中取出一个结点，计算是否有结点需要更新，如果有，并且这个结点不在队列中，那么就将它加入队列。

SPFA 算法

事实上，你会发现我们每次进行的计算可以看成是一次“松弛”。假设我们现在已经得到了 Bellman-Ford 算法某个阶段的 dist 数组，然后我们发现了一条 s 到 u 的距离比 dist[u] 更加短的路径。我们更新了 dist[u]。

那么接下来直接受到影响的就是与 u 直接关联的顶点 v，也就是如果 $\texttt{dist[u]} + \texttt{w[u][v]} < \texttt{dist[v]}$ 的话，s 到 v 的最短路就可以利用 s 到 u 的最短路加上 u 到 v 的边来更新。这样的话与 v 直接关联的顶点又会受到影响……不断这样持续下去直到最后没有顶点能被影响。

那么一个优化就是我们利用队列存储这些需要更新的结点，每次从队列中取出一个结点，计算是否有结点需要更新，如果有，并且这个结点不在队列中，那么就将它加入队列。

这样的算法被称为 SPFA——一种优化的 Bellman-Ford 算法。

SPFA 算法

在竞赛中大多数人会选择用 SPFA 作为单源最段路的算法，主要原因在于它比较好写，而且通常情况下跑得比较快。但是 SPFA 的复杂度实际上是没有保证的，最坏情况基本和 Bellman-Ford 相同，但是最好的时候可以到达 $\mathcal{O}(|\mathtt{V}|+|\mathtt{E}|)$。

SPFA 算法

在竞赛中大多数人会选择用 SPFA 作为单源最段路的算法，主要原因在于它比较好写，而且通常情况下跑得比较快。但是 SPFA 的复杂度实际上是没有保证的，最坏情况基本和 Bellman-Ford 相同，但是最好的时候可以到达 $\mathcal{O}(|V| + |E|)$。

非常重要的一点就是 SPFA 的队列需要使用**循环队列**，虽然最多队列里只会有 n 各点，但是每个点可能会入队多次。

SPFA 算法

miskcoo@bw:~/Code/oicode

```
34 void spfa()
35 {
36     std::memset(dist, 77, sizeof(dist));
37     int qhead = 0, qtail = 0;
38     dist[1] = 0, mark[1] = 1;
39     que[qtail++] = 1;
40     while(qhead != qtail)
41     {
42         int u = que[qhead++];
43         if(qhead == MaxN) qhead = 0;
44         for(int k = gp.head[u]; k; k = gp.next[k])
45         {
46             int v = gp.point[k];
47             int d = dist[u] + gp.weight[k];
48             if(d < dist[v])
49             {
50                 dist[v] = d;
51                 if(!mark[v])
52                 {
53                     mark[v] = 1;
54                     que[qtail++] = v;
55                     if(qtail == MaxN) qtail = 0;
56                 }
57             }
58         }
59
60         mark[u] = 0;
61     }
```

Dijkstra 算法

如果有向图的边权值全为正数，那么有一种复杂度有保证的单源最段路算法——Dijkstra 算法。它的复杂度是 $\mathcal{O}(|E|\log|V|)$。

Dijkstra 算法

如果有向图的边权值全为正数，那么有一种复杂度有保证的单源最段路算法——Dijkstra 算法。它的复杂度是 $\mathcal{O}(|E|\log|V|)$。

事实上，Dijkstra 算法的思想和 Prim 有很多类似之处。Dijkstra 算法维护了一个未访问的结点集合 T 以及一个从 s 到结点 u 的当前距离 dist[u]。

Dijkstra 算法

如果有向图的边权值全为正数，那么有一种复杂度有保证的单源最段路算法——Dijkstra 算法。它的复杂度是 $\mathcal{O}(|E|\log|V|)$。

事实上，Dijkstra 算法的思想和 Prim 有很多类似之处。Dijkstra 算法维护了一个未访问的结点集合 T 以及一个从 s 到结点 u 的当前距离 dist[u]。

1. 将除源外所有结点当前距离设置为 ∞，将源 s 的当前距离设置为 0，将当前节点设置为源 s。

Dijkstra 算法

如果有向图的边权值全为正数，那么有一种复杂度有保证的单源最段路算法——Dijkstra 算法。它的复杂度是 $\mathcal{O}(|E|\log|V|)$。

事实上，Dijkstra 算法的思想和 Prim 有很多类似之处。Dijkstra 算法维护了一个未访问的结点集合 T 以及一个从 s 到结点 u 的当前距离 dist[u]。

1. 将除源外所有结点当前距离设置为 ∞，将源 s 的当前距离设置为 0，将当前节点设置为源 s。
2. 从当前结点 u 开始，找出所有在未访问集合 T 中与 u 有边 (u,v) 的结点 v。如果 dist[u]+w[u][v]<dist[v]，那么就更新 dist[v] 的值。

Dijkstra 算法

如果有向图的边权值全为正数，那么有一种复杂度有保证的单源最段路算法——Dijkstra 算法。它的复杂度是 $\mathcal{O}(|E|\log|V|)$。

事实上，Dijkstra 算法的思想和 Prim 有很多类似之处。Dijkstra 算法维护了一个未访问的结点集合 T 以及一个从 s 到结点 u 的当前距离 dist[u]。

1. 将除源外所有结点当前距离设置为 ∞，将源 s 的当前距离设置为 0，将当前节点设置为源 s。
2. 从当前结点 u 开始，找出所有在未访问集合 T 中与 u 有边 (u,v) 的结点 v。如果 dist[u]+w[u][v]<dist[v]，那么就更新 dist[v] 的值。
3. 将当前节点从 T 中删除，并且找到在 T 中 dist 最小的结点设置为新的当前节点。

Dijkstra 算法

如果有向图的边权值全为正数，那么有一种复杂度有保证的单源最段路算法——Dijkstra 算法。它的复杂度是 $\mathcal{O}(|E|\log|V|)$。

事实上，Dijkstra 算法的思想和 Prim 有很多类似之处。Dijkstra 算法维护了一个未访问的结点集合 T 以及一个从 s 到结点 u 的当前距离 dist[u]。

1. 将除源外所有结点当前距离设置为 ∞，将源 s 的当前距离设置为 0，将当前节点设置为源 s。
2. 从当前结点 u 开始，找出所有在未访问集合 T 中与 u 有边 (u,v) 的结点 v。如果 dist[u]+w[u][v]<dist[v]，那么就更新 dist[v] 的值。
3. 将当前节点从 T 中删除，并且找到在 T 中 dist 最小的结点设置为新的当前节点。
4. 重复 (2) 和 (3) 直到 T 成为空集。

Dijkstra 算法

它的正确性在于，在未访问集合 T 中结点的 dist 是从 s 开始经过已经访问集合中的结点到达它的最短路。

Dijkstra 算法

它的正确性在于，在未访问集合 T 中结点的 dist 是从 s 开始经过已经访问集合中的结点到达它的最短路。

如果选出的当前结点 u 的 dist 不是最终的最小值，那么它最终的最短路一定是要经过一个此时 T 中的其它结点再到 u。这时那个结点的 dist 肯定要小于 u 的 dist，这就和 u 是 dist 最小的结点矛盾了！

Floyd 算法

如果你要求所有顶点间的最短路，当然可以对每个顶点跑单源最短路。但是，有一个更为直接的 Floyd 算法。

Floyd 算法

如果你要求所有顶点间的最短路，当然可以对每个顶点跑单源最短路。但是，有一个更为直接的 Floyd 算法。

我们设 d[k][i][j] 为除了 i 和 j 外只经过前 k 个结点，从 i 到 j 的最短路。

Floyd 算法

如果你要求所有顶点间的最短路，当然可以对每个顶点跑单源最短路。但是，有一个更为直接的 Floyd 算法。

我们设 d[k][i][j] 为除了 i 和 j 外只经过前 k 个结点，从 i 到 j 的最短路。

显然可以知道 d[0][i][j] = w[i][j]。

Floyd 算法

如果你要求所有顶点间的最短路，当然可以对每个顶点跑单源最短路。但是，有一个更为直接的 Floyd 算法。

我们设 d[k][i][j] 为除了 i 和 j 外只经过前 k 个结点，从 i 到 j 的最短路。

显然可以知道 d[0][i][j] = w[i][j]。

那么当加入了一个顶点 k 之后，最短路如果有变化的话一定是以 k 为中间顶点，那么可以得到

$$d[k][i][j] = \min(d[k-1][i][j], d[k-1][i][k] + d[k-1][k][j])$$

Floyd 算法

如果你要求所有顶点间的最短路，当然可以对每个顶点跑单源最短路。但是，有一个更为直接的 Floyd 算法。

我们设 d[k][i][j] 为除了 i 和 j 外只经过前 k 个结点，从 i 到 j 的最短路。

显然可以知道 d[0][i][j] = w[i][j]。

那么当加入了一个顶点 k 之后，最短路如果有变化的话一定是以 k 为中间顶点，那么可以得到

$$d[k][i][j] = \min(d[k-1][i][j], d[k-1][i][k] + d[k-1][k][j])$$

这个算法的复杂度是 $\mathcal{O}(|V|^3)$。

例题 1 虫洞

给出由 n 个虫洞的质量，其由 m 条单向边连接。

虫洞之间跃迁需要一定燃料和 1 个单位时间，每过 1 单位时间虫洞反色。

从白洞到黑洞，消耗的燃料减少 delta，反之燃料增加 delta (delta 为两虫洞质量差)。

可以选择在一个结点花费 s_i 的时间停留一个单位时间。

求从虫洞 1 到 n 最小的燃料消耗。

其中 $0 < n < 5000, 0 < m < 30000$。

每个点拆成黑白两个

每个点拆成黑白两个

由于每秒虫洞变色，所以黑点之间边权为 $\mathtt{v}+\mathtt{delta}$

每个点拆成黑白两个

由于每秒虫洞变色，所以黑点之间边权为 $\mathtt{v}+\mathtt{delta}$

其余连边同理

每个点拆成黑白两个

由于每秒虫洞变色，所以黑点之间边权为 $v + delta$

其余连边同理

求最短路

例题 2 BZOJ2143

给出两个 n * m 的矩阵 A，B，以及 3 个人的坐标

在 (i, j) 支付 $A_{i,j}$ 的费用可以弹射到曼哈顿距离不超过 $B_{i,j}$ 的位置

问三个人汇合所需要的最小总费用

其中 $0 < n, m < 150, 0 < A < 1000, 0 < B < 10^9$。

这道题点很少，但是边可能很多，直接建图做最短路显然不可行

这道题点很少，但是边可能很多，直接建图做最短路显然不可行

把弹射看成获得了可以走 $a_{i,j}$ 的能量

这道题点很少，但是边可能很多，直接建图做最短路显然不可行

把弹射看成获得了可以走 $a_{i,j}$ 的能量

每走一格消耗 1 的能量，$f(i,j,k)$ 表示在 (i,j) 且有 k 的能量的最少费用

这道题点很少，但是边可能很多，直接建图做最短路显然不可行

把弹射看成获得了可以走 $a_{i,j}$ 的能量

每走一格消耗 1 的能量，$f(i,j,k)$ 表示在 (i,j) 且有 k 的能量的最少费用

求三次最短路，枚举汇合点

475B.Strongly Connected City

给定 n 条水平和 m 条竖直的单向街道，其互相交叉为 n*m 个结点，问这些结点是否都能互相到达。

475B.Strongly Connected City

给定 n 条水平和 m 条竖直的单向街道，其互相交叉为 n*m 个结点，问这些结点是否都能互相到达。

$1 \leq n, m \leq 20$

475B.Strongly Connected City(1400)

题解

1. 从每个结点开始 dfs，最后看每个结点是否最终被搜到 n*m 次。

475B.Strongly Connected City(1400)

题解

1. 从每个结点开始 dfs，最后看每个结点是否最终被搜到 n*m 次。

2. 求强连通图可以直接套用 tarjan 算法。

475B.Strongly Connected City(1400)

题解

1. 从每个结点开始 dfs，最后看每个结点是否最终被搜到 n*m 次。
2. 求强连通图可以直接套用 tarjan 算法。
3. 实际上只需要判断外环路是否成环即可。

639B.Bear and Forgotten Tree 3

构造一棵 n 个点，深度 h，最长链为 d 的树。

639B.Bear and Forgotten Tree 3

构造一棵 n 个点，深度 h，最长链为 d 的树。

$1 \leq n, h, d \leq 10^5$

639B.Bear and Forgotten Tree 3(1600)

题解

首先构造一个长度为 d 的链，然后把其中一个距离边上为 h 的点变为根。

把剩下的点全都接在根上。

437C.The Child and Toy

n 个带权点，m 条边的无向图，删除一个点的代价是与这个点相邻的，且没有被删除的点权和。

437C.The Child and Toy

n 个带权点，m 条边的无向图，删除一个点的代价是与这个点相邻的，且没有被删除的点权和。

求将所有点删除的最小代价。

437C.The Child and Toy

n 个带权点，m 条边的无向图，删除一个点的代价是与这个点相邻的，且没有被删除的点权和。

求将所有点删除的最小代价。

$1 \leq n \leq 1000, 1 \leq m \leq 2000$

437C.The Child and Toy(1600)

题解

按照从大到小的次序删点。

题解

按照从大到小的次序删点。

发现每条边的贡献是连接的两个点的权值的最小值。

灾后重建

B 地区在地震过后，所有村庄都造成了一定的损毁，而这场地震却没对公路造成什么影响。但是在村庄重建好之前，所有与未重建完成的村庄的公路均无法通车。换句话说，只有连接着两个重建完成的村庄的公路才能通车，只能到达重建完成的村庄。

给出 B 地区的村庄数 n，村庄编号从 1 到 n，和所有 m 条公路的长度，公路是双向的。并给出第 i 个村庄重建完成的时间 t[i]，你可以认为是同时开始重建并在第 t[i] 天重建完成，并且在当天即可通车。若 t[i] 为 0 则说明地震未对此地区造成损坏，一开始就可以通车。之后有 Q 个询问 (x, y, t)，对于每个询问你要回答在第 t 天，从村庄 x 到村庄 y 的最短路径长度为多少。如果无法找到从 x 村庄到 y 村庄的路径，经过若干个已重建完成的村庄，或者村庄 x 或村庄 y 在第 t 天仍未重建完成，则需要返回-1。

数据保证 $t[1] \le t[2] \le \cdots \le t[n]$。并且

灾后重建

在第 i 个村庄重建完成时，前面村庄都重建完成了，后面的村庄都没有重建完成。

灾后重建

在第 i 个村庄重建完成时，前面村庄都重建完成了，后面的村庄都没有重建完成。

最短路只能经过前 i 个村庄。

灾后重建

在第 i 个村庄重建完成时，前面村庄都重建完成了，后面的村庄都没有重建完成。

最短路只能经过前 i 个村庄。

Floyd！

抗脏的道路

有一个 n 个结点 m 条边的无向图，边有权值。

从一个结点 s 到另一个结点 t 的一条路径的抗脏值定义为路径上权值最大的那条边的权值。

要求计算出从 s 到 t 的所有路径中抗脏值最小的那条路径的抗脏值。

其中 $1 \leq n \leq 10000, 1 \leq m \leq 20000$。

肮脏的道路

二分?

肮脏的道路

二分?

类似 Kruskal 算法直接判断连通。

肮脏的道路

二分?

类似 Kruskal 算法直接判断连通。

事实上，这样的路径叫做**最小瓶颈路**。

糖果

有 n 个人分糖果，现在有 m 个限制，比如说某一个人得到的糖果不能比另一个人少 k 个以上。

请你求出小 A 比小 B 最多能多得到多少个糖果。

其中 $1 \leq n \leq 30000, 1 \leq m \leq 150000$。

Source: POJ 3159. Candies

糖果

比如小 B 得到的糖果不能比小 A 少 k 个以上，那么

$$A - B \leq k$$

糖果

比如小 B 得到的糖果不能比小 A 少 k 个以上，那么

$$A - B \leq k$$

将每个人看成一个点，比如上面这个限制就从 B 向 A 连接一条长度为 k 的边。

糖果

比如小 B 得到的糖果不能比小 A 少 k 个以上，那么

$$A - B \leq k$$

将每个人看成一个点，比如上面这个限制就从 B 向 A 连接一条长度为 k 的边。

最短路

混合图

你有一个混合图，它有 n 个顶点，m 条边，其中 a 条是有向边，b 条是无向边。现在要求为这 b 条无向边确定一个方向，使得最后的有向图上不存在环。

其中 $1 \leq n, a, b \leq 10^5, m = a + b$。不用考虑无解的情况。

混合图

既然要求最后的有向图没有环，也就是要求最后是个 DAG！

混合图

既然要求最后的有向图没有环，也就是要求最后是个 DAG！

DAG 进行拓扑排序后边的方向都是拓扑序小的连向拓扑序大的。

混合图

既然要求最后的有向图没有环，也就是要求最后是个 DAG！

DAG 进行拓扑排序后边的方向都是拓扑序小的连向拓扑序大的。

先把无向边忽略，对图进行拓扑排序，最后无向边的方向就是拓扑序小的连向拓扑序大的。

NOIP2009 最优贸易

n 个点，m 条带权边的有向图，每个点有一个物品的买入价格和卖出价格

现在要从 1 号点走到 n 号点，途中会买一个物品并在这之后卖出，问最多能赚多少钱?

$1 \leq n \leq 10^5, 1 \leq m \leq 5 * 10^5$

题解

显然只要两次 spfa 处理出到某个点的路径上的最小买入价格以及其到终点路径上的最大卖出价格即可

437C.The Child and Toy

n 个带权点，m 条边的无向图，删除一个点的代价是与这个点相邻的，且没有被删除的点权和。

437C.The Child and Toy

n 个带权点，m 条边的无向图，删除一个点的代价是与这个点相邻的，且没有被删除的点权和。

求将所有点删除的最小代价。

437C.The Child and Toy

n 个带权点，m 条边的无向图，删除一个点的代价是与这个点相邻的，且没有被删除的点权和。

求将所有点删除的最小代价。

$1 \leq n \leq 1000, 1 \leq m \leq 2000$

437C.The Child and Toy(1700)

题解

按照从大到小的次序删点。

437C.The Child and Toy(1700)

题解

按照从大到小的次序删点。

发现每条边的贡献是连接的两个点的权值的最小值。

1209D.Cow and Snacks

有 n 个甜点，和 m 头牛。每头牛有两种喜欢的食物，当牛能够吃到自己喜欢的食物，就会开心，否则不开心。每只牛去吃甜点时，会把他喜欢的都吃掉。

你可以任意安排吃甜点的顺序，问至少有多少牛会伤心。

$1 \leq n, m \leq 10^5$

题解

考虑计算最多能满足多少人，把食物建点，牛建边，则每个连通块都存在一种贡献为连通块大小-1 的分配方式

1209D.Cow and Snacks(1700)

题解

考虑计算最多能满足多少人，把食物建点，牛建边，则每个连通块都存在一种贡献为连通块大小-1 的分配方式

连通块用并查集维护

1214D.Treasure Island

给一个 n * m 的字符矩阵，'.' 表示能通过，'#' 表示不能通过。每步可以往下或往右走。问至少把多少个'.' 变成'#'，才能让从 $(1,1)$ 出发不能到达 (n,m)

$1 \leq n \times m \leq 10^6$

题解

首先答案是 1 或 2，因为可以把起点直接堵上。斜着看成一个分层图，如果某一层只有一个结点和起点终点连通，答案就是 1，否则答案是 2。

题解

首先答案是 1 或 2，因为可以把起点直接堵上。斜着看成一个分层图，如果某一层只有一个结点和起点终点连通，答案就是 1，否则答案是 2。

这个判断只要从起点和终点各做一次 dp 就可以。

给定一张边权为 1 的有向图，求四个不同点 A，B，C，D 使得 dis(A, B) + dis(B, C) + dis(C, D) 取最大值

$1 \leq n \leq 3000, 1 \leq m \leq 5000$

题解

先 bfs 出任意两点的距离，$f(i,0-2)$ 表示从 i 出发的最远的三个点，$g(i,0-2)$ 表示到 i 的最远的三个点

枚举 B 和 C，再枚举 3 * 3 个点对，选出最优的 A 和 D

保留三个点的原因是保证 A，B，C，D 互不相同

复杂度 $O(n^2)$

545E.Paths and Trees

给定一个 n 个点，m 条边的无向图和一个点 u，找出若干条边组成一个子图，要求这个子图中 u 到其他点的最短距离与在原图中的相等，并且要求子图所有边的权重和最小，求出最小值。

$1 \leq n, m \leq 3 * 10^5$

求 u 到所有结点的最短路

求 u 到所有结点的最短路

记录下到每个结点最短路上的最后一条边（若有多条，取最小的）

求 u 到所有结点的最短路

记录下到每个结点最短路上的最后一条边（若有多条，取最小的）

答案是以 u 为根的一棵最短路径树

715B.Complete The Graph

给一张 n 个点，m 条边的无向图，要求设定一些边的边权使得所有边权都是正整数，最终 S 到 T 的最短路为 L

$1 \leq n \leq 1000, 1 \leq m \leq 10000$

题解

用 f(i,j) 表示从 S 到点 i，经过 j 条无边权的边的最短路

选择一个最小的 j，使得 $f(T,j)+j \le L$

题解

用 f(i,j) 表示从 S 到点 i，经过 j 条无边权的边的最短路

选择一个最小的 j，使得 $f(T,j)+j \leq L$

更改这条路径上的边权，使得最短路为 L，将其它无边权的边赋值为 L

可以证明不会产生其它的最短路

507E.Breaking Good

给定一个 n 个点，m 条边的有向图，边权都为 1，一些需要维修。

507E.Breaking Good

给定一个 n 个点，m 条边的有向图，边权都为 1，一些需要维修。

你需要选择一条 1 到 n 的最短路，将它们修好，并炸毁其它所有不在路径上的完好的路。

507E.Breaking Good

给定一个 n 个点，m 条边的有向图，边权都为 1，一些需要维修。

你需要选择一条 1 到 n 的最短路，将它们修好，并炸毁其它所有不在路径上的完好的路。

若有多条最短路，选择影响值最小的。

507E.Breaking Good

给定一个 n 个点，m 条边的有向图，边权都为 1，一些需要维修。

你需要选择一条 1 到 n 的最短路，将它们修好，并炸毁其它所有不在路径上的完好的路。

若有多条最短路，选择影响值最小的。

影响值 = 维修的路数 + 炸毁的路数。

507E.Breaking Good

给定一个 n 个点，m 条边的有向图，边权都为 1，一些需要维修。

你需要选择一条 1 到 n 的最短路，将它们修好，并炸毁其它所有不在路径上的完好的路。

若有多条最短路，选择影响值最小的。

影响值 = 维修的路数 + 炸毁的路数。

$1 \leq n, m \leq 10^5$

题解

维修的路数 = 最短路长度-最短路上完好的路数

题解

维修的路数 = 最短路长度-最短路上完好的路数

炸毁的路数 = 总完好路数-最短路上完好的路数

题解

维修的路数 = 最短路长度-最短路上完好的路数

炸毁的路数 = 总完好路数-最短路上完好的路数

所以只要选择一条完好的路最多的最短路

题解

维修的路数 = 最短路长度-最短路上完好的路数

炸毁的路数 = 总完好路数-最短路上完好的路数

所以只要选择一条完好的路最多的最短路

分层图 bfs

543B.Destroying Roads

n 个结点,m 条边的无向图 (边权全为 1)，问最多能删掉多少条边使得 s1 到 t1 距离不超过 l1,s2 到 t2 距离不超过 l2。

543B.Destroying Roads

n 个结点,m 条边的无向图 (边权全为 1)，问最多能删掉多少条边使得 s1 到 t1 距离不超过 l1,s2 到 t2 距离不超过 l2。

$1 \leq n \leq 500, 1 \leq m \leq n(n-1)/2$

题解

其实就是问，至少需要多少条边，才能使得 s1 到 t1 距离不超过 l1,s2 到 t2 距离不超过 l2。

题解

其实就是问，至少需要多少条边，才能使得 s1 到 t1 距离不超过 l1,s2 到 t2 距离不超过 l2。

如果这两条路径不相交，那么答案为 dis(s1,t1)+dis(s2,t2)。

题解

其实就是问，至少需要多少条边，才能使得 s1 到 t1 距离不超过 l1,s2 到 t2 距离不超过 l2。

如果这两条路径不相交，那么答案为 dis(s1,t1)+dis(s2,t2)。

如果相交部分为 (p1,p2)，答案为 p1,p2 的最短路，加上 p1,p2 到其它 4 个点的最短路。

题解

其实就是问，至少需要多少条边，才能使得 s1 到 t1 距离不超过 l1,s2 到 t2 距离不超过 l2。

如果这两条路径不相交，那么答案为 dis(s1,t1)+dis(s2,t2)。

如果相交部分为 (p1,p2)，答案为 p1,p2 的最短路，加上 p1,p2 到其它 4 个点的最短路。

预处理任意点对距离，枚举两条路径重叠部分。

Usaco2012Jan.Bovine Alliance

给出 n 个点 m 条边的图，现把点和边分组，每条边只能和相邻两点之一分在一组，点不可以和多条边一组，但点可以单独一组，问分组方案数。

答案对 10^9+7 取模

$1 \le n, m \le 10^5$

题解

连通块是独立的，设某个连通块点数为 n，边数为 m

题解

连通块是独立的，设某个连通块点数为 n，边数为 m

若 m>n，边数大于点数，必然有一个点要与多条边分在一组，无解

题解

连通块是独立的，设某个连通块点数为 n，边数为 m

若 m>n，边数大于点数，必然有一个点要与多条边分在一组，无解

若 m=n，环 + 外向树，解为 2

题解

连通块是独立的，设某个连通块点数为 n，边数为 m

若 m>n，边数大于点数，必然有一个点要与多条边分在一组，无解

若 m=n，环 + 外向树，解为 2

若 m=n-1，树，解为 n

题解

连通块是独立的，设某个连通块点数为 n，边数为 m

若 m>n，边数大于点数，必然有一个点要与多条边分在一组，无解

若 m=n，环 + 外向树，解为 2

若 m=n-1，树，解为 n

若 m<n-1，不存在

题解

连通块是独立的，设某个连通块点数为 n，边数为 m

若 m>n，边数大于点数，必然有一个点要与多条边分在一组，无解

若 m=n，环 + 外向树，解为 2

若 m=n-1，树，解为 n

若 m<n-1，不存在

使用并查集维护连通块的边数和点数

冷战

给定 n 个点的图。动态的往图中加边，并且询问某两个点最早什么时候联通，强制在线。

$1 \leq n \leq 10^5$

题解

考虑并查集。并查集实际上维护了一棵树。那么假如我们按秩合并，这棵树的深度是 O(log n) 的。

题解

考虑并查集。并查集实际上维护了一棵树。那么假如我们按秩合并，这棵树的深度是 $O(\log n)$ 的。

我们将一个点连向其父亲的边权设为这条边加入的时间，那么每次询问时，暴力查询树上从 u 到 v 所经过边权的最大值即可。

题解

考虑并查集。并查集实际上维护了一棵树。那么假如我们按秩合并，这棵树的深度是 O(log n) 的。

我们将一个点连向其父亲的边权设为这条边加入的时间，那么每次询问时，暴力查询树上从 u 到 v 所经过边权的最大值即可。

时间复杂度为 O(nlogn)，常数较小。

链型网络

给定一张无重边、自环的无向图.

每次可以加边，或者询问有多少个点满足将该点删除后，原图的每个连通块都为一条链

$1 \leq \mathtt{n}, \mathtt{m} \leq 10^5$

直观感受

对于下面一些简单的情况，我们可以容易得出答案.

- 原图为若干条链，则答案为点数 N;
- 原图为单个简单环加若干条链，则答案为环大小;
- 原图中超过一个连通块有环，答案为 0.
- 原图中一个连通块为一个点连接三条链，答案为 4;
- 原图中一个连通块为一个点连接三条以上的链，答案为 1;

但是考虑到环套树这样更复杂的情况，上述分类讨论也无能为力.

但是考虑到环套树这样更复杂的情况，上述分类讨论也无能为力.

我们需要思考链本身的性质，一个图的每个连通块为链，等价于每个点的度数小于等于 2 且无环. 转化后的条件明显更有利于解决问题.

首先考虑图中是否有度数大于等于 3 的点，如果存在一个度数大于等于 3 的点 u，因为要保证去掉一个点后每个点的度数都小于等于 2，我们要么去掉 u，要么在 u 度数恰好为 3 时，去掉 u 的三个相邻点中的一个．而其他的点均不可能成为答案．

这样我们只需要在加边过程中第一次出现度数为 3 的点的时候，对于可能成为答案的 4 个点，分别维护一个去掉该点之后的图.

然后在 4 个图中分别进行判断

只要在加边时判断每个点度数都小于等于 2，再用并查集判断是否有环即可.

剩下的就是每个点的度数都小于等于 2 的情况，这样每个连通块只能是链或者简单环，我们只需采用一开始的分类讨论即可.

- 原图为若干条链，则答案为点数 N;
- 原图为单个简单环加若干条链，则答案为环大小;
- 原图中超过一个连通块有环，答案为 0.

这只需要维护一个记录集合大小的并查集就能做到.